Remarks

Claims 1-7, 9-20, and 24-27 were pending as of the action mailed on September 11, 2007. Claims 1, 13, 17, and 25 are in independent form.

Claims 1, 6, 7, 9, 10, 11, and 12 are being amended. No new matter has been added.

Reexamination of the application and reconsideration of the action are respectfully requested in light of the foregoing amendments and the following remarks.

Section 102 Rejections

Claims 1-6, 9-19, and 24-27 were rejected under 35 U.S.C. § 102(e) as allegedly being fully anticipated by U.S. Patent No. 6,862,694 ("Tormey").

Claims 1, 13, 17, and 25

Claim 1 recites that checkpoints are "defined by a respective statement in source code of the computer program." The examiner points to Tormey as teaching that "each checkpoint in the plurality of checkpoints being defined by a respective statement in source code of the computer program (see for example page 4, lines 25-47, 'address')." The applicant disagrees. In fact, page 4, lines 25-47 in Tormey reads as follows:

> The CC is connected to the physical memory 13, and includes mapping tables 17 which map physical addresses received from the Cache to memory locations. A User Interface (UI) 18 is also present, either on, but preferably off, the CPU. The UI communicates with the Cache and other system components. A virtual-to-physical mapping table 19 in the CPU receives breakpoint information from the UI and converts virtual addresses to physical addresses which are passed to the Cache for storage in the memory 13.
>
> FIG. 2 is a flow chart of the steps of the conventional method of setting up a plurality of breakpoints. With reference to FIGS. 1 and 2, the method starts at step 21 where the User Interface (UI) 18 receives a virtual address of a breakpoint from the user. At step 22, it is determined whether the virtual-to-physical mapping table 19 is set up. If not, the method moves to step 29, and the process fails. However, if the virtual-to-physical mapping table is set up, the UI obtains the physical address of the breakpoint from the table at step 23. At step 24, the UI sends the physical address of the breakpoint to the Instruction Cache 14, and at step 25, the Cache sends the physical address to the Coherency Controller (CC) 16.

The cited passage of Tormey does not disclose any source code nor does it disclose checkpoints defined in the source code of the computer program that is being debugged. Tormey discloses breakpoint information entered into a table through a user interface. The breakpoints in the table convert virtual addresses to physical addresses and are used to debug a separate program code. Accordingly, the addresses stored in the table disclosed in Tormey are not checkpoints "defined by a respective statement in source code of the computer program" because the addresses stored in the table are not part of the source code of the computer program that is being debugged, which is what the claim recites.

The examiner also points to Tormey as teaching that each checkpoint is assigned to a checkpoint group by "the statement defining the respective checkpoint (see for example page 4, lines 25-47, 'table')." Page 4, lines 25-47 has been reproduced above. This passage does not disclose any source code nor does it disclose assigning each checkpoint to a checkpoint group in the source code of the computer program that is being debugged. The table disclosed in Tormey contains breakpoints that are used to debug a separate program code. Accordingly, the table in Tormey does not assign each checkpoint to a checkpoint group through the statement in the source code of the computer program "defining the respective checkpoint" because the table is not part of the source code of the computer program that is being debugged.

Additionally, because a statement in the source code of a computer program defines each checkpoint, the checkpoint executes at the location of the statement that defines it in the source code. In contrast, Tormey discloses a system and method for executing breakpoints when the computer program code requests a memory address listed in a breakpoint table. Tormey, Col. 3, ll. 24-30. The breakpoints of Tormey are defined from the outside, and do not exist within the program that is be debugged, as recited in the claim. Tormey's breakpoints are executed based on addresses stored in a table separate from the program code, and therefore do not meet the limitations of the claim.

For the foregoing reasons, the rejection of claim 1 and its dependent claims should be withdrawn.

The rejections of claims 13, 17, and 25 correspond to the rejection of claim 1. For the reasons set forth above in reference to claim 1, the rejections of claims 13, 17, and 25 should also be withdrawn.

Claims 2, 14, and 24

Claim 2 recites that "checkpoints comprise assertion statements and breakpoint statements." Assertion statements test "whether a specified condition is true or false." Specification, p.1, ll. 6, 21-22. Breakpoint statements, on the other hand, "halt the execution of a computer program." Specification, p.1, l. 5.

The examiner points to Tormey as teaching "checkpoints comprise assertion statements and breakpoint statements (see for example Abstract)." The applicant disagrees. In fact, the abstract of Tormey reads as follows:

> A system and method for setting and executing breakpoints utilized for debugging program code. A user interface (UI) stores breakpoint addresses in a breakpoint table within a central processing unit (CPU). Multiple breakpoint addresses may be stored in the table as a range of addresses in a single entry. A flag indicates whether each stored address or address range is a physical or virtual address. When executing the program code on the CPU, an instruction core requests from an instruction cache, an instruction associated with a particular address. The cache first determines from the breakpoint table within the CPU whether there is a breakpoint associated with the particular address. If so, the cache returns control to the UI. Otherwise, the cache goes out to a coherency controller to fetch the instruction from memory.

The abstract of Tormey only discloses breakpoints that halt the program execution and return control to the user interface. Nothing in the cited text discloses testing whether a specified assertion condition is true or false as required by the claim.

For these additional reasons, the rejections of claim 2 and its dependent claim should be withdrawn.

The rejections of claims 14 and 24 correspond to the rejection of claim 2. For the reasons set forth above in reference to claim 2, the rejections of claims 14 and 24 should also be withdrawn.

Claim 3

Claim 3 recites that activation variants "enable multiple checkpoint groups to be managed jointly." An activation variant is a collection of values used "to parameterize the process of activating checkpoints with regard to a set of checkpoints to be activated." Specification, p. 3, ll. 20-21.

The examiner points to Tormey as teaching "activation variants to enable multiple checkpoint groups to be managed jointly (see for example Fig. 4, item 16, 'Coherency Controller', and related text)." The applicant disagrees. In fact, the related text of coherency controller is found in Col. 7, l. 49 through Col. 8, l. 8 and reads as follows:

> However, if there was not an address match, the method moves to step 79 where the Instruction Cache 14 sends a request over the bus 15 to the CC 16 to fetch the instruction from memory 13. The method then moves to FIG. 6B, step 81, where it is determined whether the CC mapping tables 17 are set up in the CC. If not, the method moves to step 84, and the process fails. However, if the CC mapping tables are set up, the method moves to step 82 where the CC determines the memory location of the instruction from the CC mapping tables, and requests the contents of the memory at the mapped address. At step 83, it is determined whether the memory has backing. If not, the method moves to step 84, and the process fails. However, if the memory has backing, the method moves to step 85 where the memory responds by sending the stored instruction to the CC. The retrieved instruction is then returned to the Instruction Cache 14 at step 86. At step 87, the Cache sends the retrieved instruction to the Instruction Core 12 which executes it at step 88.
>
> It should be noted that the Instruction Cache 14 goes out to the CC 16, through the CC mapping tables 17, and to memory 13 to fetch the instruction only if it is first determined that there is no breakpoint associated with the address. During execution, the method determines from the Breakpoint Table 61 in the CPU 11 whether there is an associated breakpoint. This is much more efficient than the old method which had to go over the bus to the CC, through the CC mapping tables, and to memory and back just to determine whether there was an associated breakpoint.

The coherency controller determines the memory location of a computer program instruction and requests the contents from that memory location. Nothing in the cited text or

from the cited figure discloses managing groups of breakpoints jointly, whether by establishing activation variants or otherwise.

For the foregoing additional reason, the rejection of claim 3 should be withdrawn.

Claim 4

Claim 4 recites that the product includes instructions to "activate the checkpoints in the first checkpoint group." The examiner points to Tormey as teaching instructions to "activate the checkpoints in the first checkpoint group (see for example Fig. 4, item 16, 'Coherency Controller', and related text)." The related text of coherency controller has been reproduced above. As can be readily seen, nothing in the cited text or from the cited figure discloses activating checkpoints in a checkpoint group, as expressly recited in the claim.

For the foregoing additional reason, the rejection of claim 4 should be withdrawn.

Claim 5

Claim 5 recites "checkpoints that are assertions." As discussed above in reference to claims 2, 14, and 24, Tormey does not disclose the use of assertions, which tests whether a specified condition is true or false.

For the foregoing additional reason, the rejection of claim 5 should be withdrawn.

Claim 15

Claim 15 recites a "means for associating an activation variant with a checkpoint group." The examiner points to Tormey as teaching a "means for associating an activation variant with a checkpoint group (see for example Fig. 4, item 16, 'Coherency Controller', and related text)."

This rejection corresponds to the rejection of claim 3.

For the reasons set forth above in reference to claim 3, the rejection of claim 15 should also be withdrawn.

Claim 16

Claim 16 recites a "means for associating an activation variant with a compilation unit." The examiner points to Tormey as teaching a "means for associating an activation variant with a compilation unit (see for example Fig. 4, item 16, 'Coherency Controller', and related text)." The related text of coherency controller has been reproduced above. As can be readily seen,

nothing in the cited text or from the cited figure discloses associating an activation variant with a compilation unit. In fact, there is no mention of or reference to a compilation unit in the cited passage.

For the foregoing additional reason, the rejection of claim 16 should be withdrawn.

Remaining Claims

The remaining claims depend from or correspond to independent claims 1, 13, 17, and 25 and are allowable for at least the reasons that apply to independent claims 1, 13, 17, and 25.

Withdrawal of the rejection under 35 U.S.C. § 102(e) is therefore respectfully requested.

Section 103 Rejections

Claims 7 and 20 were rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over Tormey in view of U.S. Patent Publication No. 2003/0217354 A1 ("Bates").

Claim 7

Claim 7 depends from claim 1 and is allowable for at least the reasons that apply to claim 1, as discussed above.

Additionally, claim 7 recites that checkpoint activation is "performed only for a particular server of multiple servers." The examiner points to Bates as teaching "a control input specifying that activating is to be performed only for a particular server of multiple servers on which the first computer program is running (see for example, page 2, [0032], and Fig. 2, and related text)." The applicant disagrees. In fact, page 2, [0032] reads as follows:

> [0032] Referring now to FIG. 1, a computing environment 100 is shown. In general, the distributed environment 100 includes a computer system 110 and a plurality of networked devices 146. For simplicity, only the details of the computer system 110 are shown. However, it is understood that the computer system 110 may be representative of one or more of the networked devices 146. In general, computer system 110 and the networked devices 146 could be any type of computer, computer system or other programmable electronic device, including desktop or PC-based computers, workstations, network terminals, a client computer, a server computer, a portable computer, an embedded controller, etc.

The related text for Fig. 2 in Bates corresponds to p.3, [0040] – [0043], which read in their entirety as follows:

> [0040] An illustrative debugging process is now described with reference to FIG. 2. A debugging process is initiated by the debug user interface 124. The user interface 124 presents the program under debugging and highlights the current line of the program on which a stop or error occurs. The user interface 124 allows the user to set control points (e.g., breakpoints and watch points), display and change variable values, and activate other inventive features described herein by inputting the appropriate commands. In some instances, the user may define the commands by referring to high-order language (HOL) references such as line or statement numbers or software object references such as a program or module name, from which the physical memory address may be cross-referenced.
>
> [0041] The expression evaluator 126 parses the debugger command passed from the user interface 124 and uses a data structure (e.g., a table) generated by the compiler 120 to map the line number in the debugger command to the physical memory address in memory 116. In addition, the expression evaluator 126 generates a Dcode program for the command. The Dcode program is machine executable language that emulates the commands. Some embodiments of the invention include Dcodes which, when executed, activate control features described in more detail below.
>
> [0042] The Dcode generated by the expression evaluator 126 is executed by the Dcode interpreter 128. The interpreter 128 handles expressions and Dcode instructions to perform various debugging steps. Results from Dcode interpreter 128 are returned to the user interface 124 through the expression evaluator 126. In addition, the Dcode interpreter 128 passes on information to the stop handler 134, which takes steps described below.
>
> [0043] After the commands are entered, the user provides an input that resumes execution of the program 120. During execution, control is returned to the debugger 123 via the stop handler 134. The stop handler 134 is a code segment that returns control to the appropriate user interface. In some implementations, execution of the program eventually results in an event causing a trap to fire (e.g., a breakpoint or watchpoint is encountered). Inserting and managing special op codes that cause these traps to fire is the responsibility of the breakpoint manager 130. When a trap fires, control is then returned to the debugger by the stop handler 134 and program execution is halted. The stop handler 134 then invokes the debug user interface 124 and may pass the results to the user interface 124.

> Alternatively, the results may be passed to the results buffer 136 to cache data for the user interface 124. In other embodiments, the user may input a command while the program is stopped, causing the debugger to run a desired debugging routine. Result values are then provided to the user via the user interface 124.

Bates discloses a debugging process in a distributed environment initiated by a user interface. Nothing in the cited text or cited figure discloses activating checkpoints only for a particular server.

For the foregoing additional reason, the rejection of claim 7 should be withdrawn.

Claim 20

The rejection of claim 20 corresponds to the rejection of claim 7.

For the reasons set forth above in reference to claim 7, the rejection of claim 20 should also be withdrawn.

Conclusion

The applicant respectfully requests that all pending claims be allowed.

By responding in the foregoing remarks only to particular positions taken by the examiner, the applicant does not acquiesce with other positions that have not been explicitly addressed. In addition, the applicant's selecting come particular arguments for the patentability of a claim should not be understood as implying that no other reasons for the patentability of that claim exist. Finally, the applicant's decision to amend or cancel any claim should not be understood as implying that the applicant agrees with any positions taken by the examiner with respect to that claim or other claims.

Please apply any charges not otherwise paid or any credits to Deposit Account No. 06-1050.

Respectfully submitted,

Date: December 5, 2007

Brian Gustafson Reg. No. 52,978 For,
Hans R. Troesch
Reg. No. 36,950

Customer No. 32864
Fish & Richardson P.C.
Telephone: (650) 839-5070
Facsimile: (650) 839-5071